IOT-BASED SMART PARKING SYSTEM

PHASE 3

DEVELOPMENT PART 1

Developing a smart parking project using iot (internet of things) involves integrating sensors, microcontrollers, communication modules, and a software platform to efficiently manage parking spaces.

COMPONENTS FEATURES:

HARDWARE COMPONENTS:

Sensors:

sensors that use infrared light or ultrasonic waves to detect the presence of cars in parking spaces.we use infrared sensors in our project

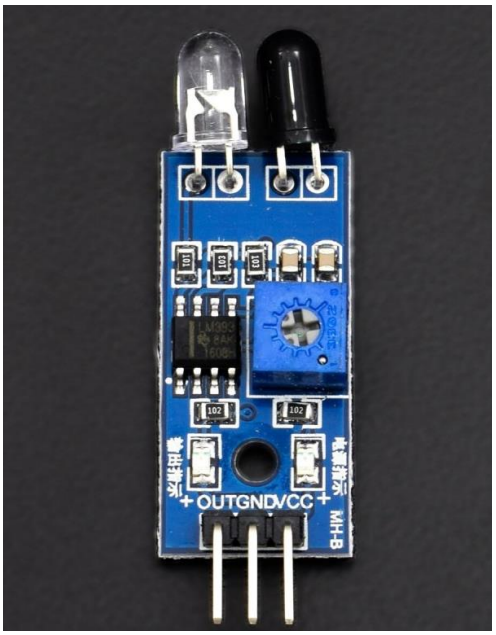

Figure 1 infrared sensors

Features:

- identifies whether or not cars are present in parking spots.
- provides information on occupancy status.

Microcontrollers:

You may connect to the sensors using an arduino or raspberry pi and upload data to the cloud.we use raspberry-pi in our project.

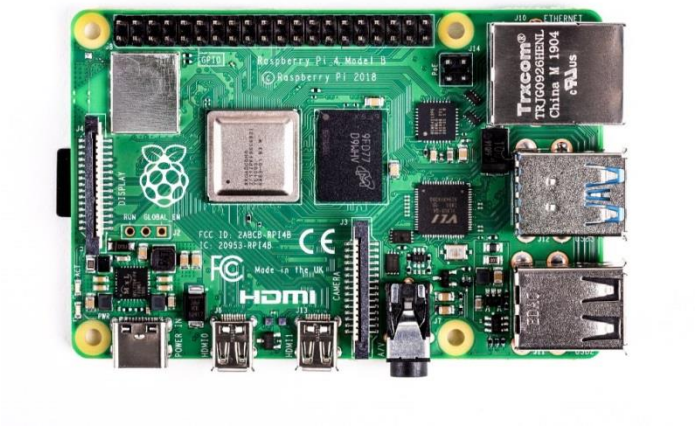

Figure 2: raspberry pi

Features:

- Handles sensor data processing.
- Maintains and controls the hardware parts.

Modules for communication:

modules for wi-fi, bluetooth, or lora to relay data to the main server.

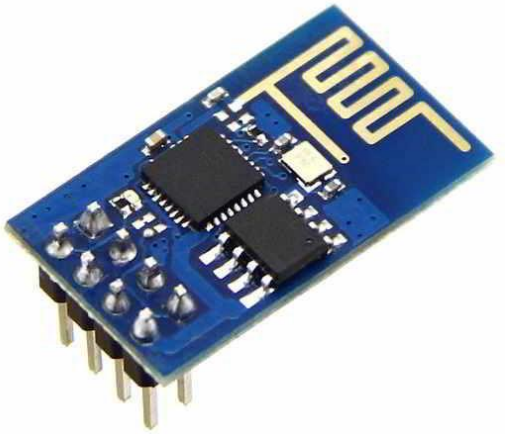

Figure 3: wi-fi module

Features:

- Sends sensor data to a central server or cloud computing platform.
- Allows for in-the-moment conversation.ponents.

Display units:

Types:

Led displays, lcd screens, or mobile apps.we use lcd screen in our our project

Features:

- Provides real-time information on parking availability to users.

Figure 4: lcd display

Barrier/gate control:

we servo motor used for barrier in our project

Figure 5: servo motor

Features:

- Enables automated entry and exit of vehicles based on availability and user permissions.
- Controlled by the central system.

SOFTWARE COMPONENTS:

Web or mobile app user interface:

- Allows customers to reserve parking spaces and view available spaces.
- Contains details on location, user account administration, and parking charges.

API for Parking Availability:

- Gives current information on the availability of parking spaces.
- Supports both occupied and vacant spot searches.

Reserving Method:

- Enables customers to secure parking spaces ahead of time.
- Upon successful reservation, sends emails or alerts of confirmation.

Data processing for sensors:

- Gathers information from sensors, such as ultrasonic sensors, to estimate the availability of parking spaces.
- Uses algorithms to analyze sensor data and modify the status of availability.

DBMS, or database management system

- Keeps records of user accounts, reservations, and parking places.
- Enables effective data management and retrieval.

Processing of payments (Optional):

- Allows users to pay for parking services online by integrating a payment gateway.
- Safely handles transactions

TO CREATE A SMART PARKING SYSTEM USING IOT DEVICES AND PYTHON SCRIPTS, YOU'LL NEED TO FOLLOW A SERIES OF STEPS:

Step 1: define requirements and components:

Define requirements: clearly state the features you want your smart parking system to have in the requirements you define. For instance, you could wish to keep track of occupancy levels, identify the presence of a car in a parking space, and notify users in real time.

Select components: decide on the hardware elements, such as sensors, microcontrollers, and communication modules, that are required.

Step 2: set up hardware:

Connect sensors: make sure the chosen sensors are powered and linked properly before wiring them up to your microcontroller (such as an arduino, raspberry pi, etc.).

Step 3: write python script for iot devices:

Installing required libraries: you might need to install particular libraries for interacting with hardware components depending on the sensors and microcontroller you're using. For instance, rpi.gpio or other sensor-specific libraries could be required if you're using a raspberry pi.

Code for vehicle detection: for the purpose of determining if a parking space is occupied or not, create a python script that receives data from the sensors. This can entail establishing interruptions or periodically polling the sensor.

Communication logic: create a channel for communication between your iot devices and a main server or cloud platform. Any acceptable protocol, including http and mqtt, can be used for this.

Handle edge cases: when writing your script, include error handling and take into account eventualities like sensor failure, network problems, or power outages.

Step 4: implement data transmission

Data sending: you might need to communicate data in real-time to a central server or a cloud platform, depending on the specifications of your project. This can entail establishing a websocket connection, using API, or utilizing MQTT brokers.

Data storage (optional): implementing a storage solution like a database may be necessary if you wish to do data analytics or log past data.

Step 5: develop centralized server or cloud platform

Set up server/cloud: to receive and handle data from your iot devices, you should either create a central server or utilize a cloud platform like aws, google cloud, or azure.

Data processing: develop the necessary backend logic to receive, process, and store data from the iot devices.

Step 6: user interface

Create a ui: create a user interface that will allow users to obtain parking information. A online or mobile application that connects to your server or cloud platform may be this.

Step 7: testing and deployment

Testing: to make sure your system performs as planned, thoroughly test all of its components as well as the whole integrated system.

Deployment: place your iot devices in the parking lot, making sure they are adequately powered and networked.

Step 8: maintenance and monitoring

Monitoring: implement monitoring systems to track the performance of your iot devices and the server/cloud platform. Set up alerts for any anomalies.

Maintenance: regularly maintain the hardware and software components, including firmware updates, sensor calibration, and addressing any issues that arise.

A Raspberry Pi, an IR sensor, a servo motor, a WiFi module, an LCD display, and a mobile application are used to build a smart parking system. To get you started, consider the simplistic example below. Be aware that the Raspberry Pi script is the main emphasis of this demonstration. To finish the system, you must build a server and a mobile application.

Raspberry Pi Script:

```
import RPi.GPIO as GPIO

import time

import serial

from RPLCD import CharLCD


# Define GPIO pins

IR_SENSOR = 18

SERVO = 12

GREEN_LED = 17

RED_LED = 27


# Define WiFi module serial port

WIFI_SERIAL_PORT = "/dev/ttyUSB0"


# Initialize LCD

lcd = CharLCD(numbering_mode=GPIO.BCM, cols=16, rows=2, pin_rs=26, pin_e=19, pins_data=[13, 6, 5, 11])


# Initialize WiFi module

wifi_serial = serial.Serial(WIFI_SERIAL_PORT, baudrate=9600, timeout=1)


# Set up GPIO mode and pins

GPIO.setmode(GPIO.BCM)

GPIO.setup(IR_SENSOR, GPIO.IN)

GPIO.setup(SERVO, GPIO.OUT)
```

```
GPIO.setup(GREEN_LED, GPIO.OUT)
GPIO.setup(RED_LED, GPIO.OUT)

def is_parking_spot_vacant():
    return GPIO.input(IR_SENSOR) == GPIO.HIGH

def move_servo(angle):
    pwm = GPIO.PWM(SERVO, 50)
    pwm.start(0)
    duty_cycle = 2 + (angle / 18)
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(0.5)
    pwm.stop()

def update_leds(parking_status):
    if parking_status == "vacant":
        GPIO.output(GREEN_LED, GPIO.HIGH)
        GPIO.output(RED_LED, GPIO.LOW)
    elif parking_status == "occupied":
        GPIO.output(GREEN_LED, GPIO.LOW)
        GPIO.output(RED_LED, GPIO.HIGH)

def update_lcd(message):
    lcd.clear()
    lcd.write_string(message)

def send_wifi_command(command):
    wifi_serial.write((command + '\r\n').encode())
    response = wifi_serial.readline().decode()
    return response.strip()
```

```
try:
    while True:
        if is_parking_spot_vacant():
            update_leds("vacant")
            move_servo(90)
            update_lcd("Parking Vacant")
            send_wifi_command("PARKING_VACANT")
        else:
            update_leds("occupied")
            move_servo(0)
            update_lcd("Parking Occupied")
            send_wifi_command("PARKING_OCCUPIED")

        time.sleep(2)

except KeyboardInterrupt:
    GPIO.cleanup()
    wifi_serial.close()
```

In this script:

- The IR sensor's input is used to determine if the parking spot is vacant or occupied.
- The servo motor is controlled based on the parking status.
- The LCD display is updated accordingly.
- The wifi module is used to send parking status information.

Here are some important details as follows:

- The pin numbers need be adjusted to properly connect your components (IR sensor, servo motor, wifi module, and LCD).

- Pip install rpi.GPIO rplcd can be used to install the required libraries.

- To match the serial port on your particular wifi module, adjust the WIFI_SERIAL_PORT variable.

- According to your hardware needs, use the update_leds(), move_servo(), and send_wifi_command() routines.

- The wifi module is assumed to adhere to a straightforward command-response protocol in this script. Adapt the send_wifi_command() method to the real wifi module protocol.